软件定义网络中利用 IMKVS 结合 NFV 的分布式网络负载均衡策略 *

钟百胜 [1]，姜利群 [2]

(1. 广州工商学院 计算机科学与工程系，广州 510800; 2. 中国矿业大学 计算机学院，江苏 徐州 221116)

摘　要：社交网络和其他云应用程序应该能对从数据中心发出的请求作出快速响应，实现这种请求的技术之一是内存中的键值存储（IMKVS），它是一种缓存机制，目的是为了提高整体用户体验。一般地，IMKVS 系统使用一致性哈希来决定在哪存储目标，一致性哈希使用起来方法简单，但可能引起网络负载的不平衡。为了提高 IMKVS 的缓存性能，提出一种软件定义网络中利用 IMKVS 结合 NFV 的分布式网络负载均衡策略。该策略包含两个阶段，第一阶段设计通用的 SDN 负载平衡器的模块，以运行不同的负载平衡算法；第二阶段是基于 IMKVS 的专业化缓存，可以实现通信管理和数据复制。仿真结果表明，相比于一致性哈希，缓存服务器上的负载可改善 24%，网络上的负载可改善 7%，策略能够使资源利用更合理，获得更好的用户体验。

关键词：软件定义网络；内存中的键值存储；网络功能虚拟化；负载均衡；缓存；应用层流量优化

中图分类号：TP393.07　　doi: 10.3969/j.issn.1001-3695.2018.01.0001

Distributed network load balancing strategy using IMKVS and NFV in software defined networks

Zhong Baisheng[1], Jang Liqun[2]

(1. *Dept of Computer Sciences & Engineering*, *Guangzhou College of Technology & Business*, *Guangzhou* 510800, *China*; 2. *College of Computer Science*, *China University of Mining & Technology*, *Xuzhou Jiangsu* 221116, *China*)

Abstract: Social networks and other clouding applications should require fast responses from datacenter are infra- structure. One of the techniques that have been widely used for achieving such requirement is the employment of In-Memory Key-Value Storage (IMKVS) as caching mechanisms in order to improve overall user experience. Commonly IMKVS systems use Consistent Hashing to decide where to store an object, which may cause network load imbalance due to its simplistic approach. In order to improve IMKVS performance, this paper proposed a distributed network load balancing strategy using IMKVS and NFV in software defined networks. The strategy consisted of two phases. In the first phase, a generic SDN load balancer module was designed to run different load balancing algorithms. The second phase was a specialized caching based on IMKVS that enabled communication management and data replication. Simulation results show an improvement of 24% on the load of the caching servers and 7% on the load of the network compared to consistent hashing, which results in better resource usage and better user experience.

Key words: software defined networks; in-memory key-value storage; network function virtualization; load balancing; cache; application-layer traffic optimization

0　引言

随着社交网络的日益普及和云应用程序的日益复杂，建立一个可以处理强大数据的数据中心是非常有必要的。为了支持大量数据的存储和处理，许多云应用程序采用一个简单而有效的缓存基础设施，该设施依赖于内存中的键值存储（in-memory key-value stores，IMKVS）[1]。这些简单存储可以通过密钥映射为任意类型的数据提供快速的访问。IMKVS 常用以存储访问的数据库查询或复杂的计算与结果等。文献[2,3]分别介绍了在大型云服务中已经开发和部署的 IMKVS 缓存实现。文献[4]的应用程序编程接口（application programming interface，API）客户端使用了一致性哈希（consistent hash，CH）[5]技术。它包含在

收稿日期：2018-01-04；**修回日期：**2018-02-09　**基金项目：**国家自然科学基金资助项目（61762086）；广东省青年创新人才类基金项目（2015KQNCX196）

作者简介：钟百胜（1982-），男，广东五华人，实验师，硕士，主要研究方向为计算机网络等（wu82142song4210@163.com）；姜利群（1956-），女，江苏海门人，副教授，硕导，主要研究方向为计算机应用等.

服务器之间均匀分布密钥，确保服务器的添加和移除均不会对密钥分布产生巨大影响。尽管 CH 是一种非常有效的技术，但它实际上是一种特殊的散列函数，该函数使用密钥和一组服务器来选择在哪存储数据。然而，它可能在生产网络中出现负载不平衡，因为它不考虑环境方面和对象特征，像尺寸、链路拥塞或对象普及度。

此外，在数据中心环境中，有必要通过应用层算法或网络业务流程实现网络和服务器负载平衡。事实上，在过去的几年里，找到专门的硬件设备或应用程序能够执行特定类型服务的流量负载平衡是很常见的。然而，负载均衡任务不应被耦合到专门的基础设施项目中，因为为了确保最大性能，它应是网络本身的一个嵌入式特征。

在过去的几年里，网络功能虚拟化（network function virtualization，NFV）[6]已成为现代计算机网络技术最有前途的研究领域之一。使用旨在替换运行网络功能的专有硬件设备的软件和虚拟化，如域名服务(DNS)、网络地址转换(NAT)、入侵检测系统(IDS)、缓存等，NFV 带来了开发网络服务的新方法。在 NFV 中，通过软件实施这些称作虚拟化网络功能(virtualization network function，VNF)的服务器，并将其部署于虚拟机中(VM)。

与 NFV 的同时，另一种越来越受欢迎的网络方法为软件定义网络（software defined network，SDN）[7]。通过从数据平面中分离控制平面，SDN 通过创建网络低级功能的抽象化概念来管理网络。尽管 SDN 和 NFV 具有许多共同的方面，但是它们既非竞争者，又不冲突。当一起使用它们时，整个网络倾向于从中受益，因为它具有与虚拟化电源并存的数据和控制平面分离的最佳方案，从而对网络进行了有效的控制。SDN 有助于促进更好的通信业务流程，而 NFV 集中于业务提供。

本文研究旨在提出一种可以平衡 IMKVS 请求负载的新 VNF，它集中于数据中心环境特征。为了最小化数据中心网络和服务器的负载，从而提供高的可扩展性、更好的资源使用和复制机制以减轻由缓存层中流行对象产生的负载，这种新的 VNF 将考虑各种网络和服务器的功能。除此之外，为了避免创建网络瓶颈，本文将提出一种通用的基于 ALTO 的负载平衡，并将其充当 VNF 的前端代理。

1　相关研究

文献[8]提出了一种负载平衡方法，它通过将 IP 地址空间切割为孤立到服务器端和客户端的树，旨在主动地平衡从客户端到服务器端的通信负载。该方法提出广泛使用通配符，通配符可以减少转发性能并创建管理问题。文献[9]提出了 Plug-n-Server，它可以平衡非结构化网络上的负载，旨在最小化 HTTP 服务器的平均响应时间。Plug-n-Server 通过聚集有关 CPI 的度量和网络链路上的网络拥塞来平衡 HTTP 请求负载，它使用负载平衡算法来选择合适的服务器来引导请求，同时控制了网络上的数据包所采用的路径。文献[10]提出了网络辅助查找方法，它是一种通过现有 IP 基础设施来对关键值存储进行快速负载平衡的方法。其提议的解决方案包含分配多个 IP 地址到每层的每个服务器，每个地址被映射到很多对象。

2　关键技术

2.1　软件定义网络和应用层流量优化协议

SDN 是网络控制和管理的一种方法，该方法允许管理员通过较低网络协议功能的抽象化来管理网络服务。这是通过解耦控制平面来完成的，它决定了通过网络基础结构而转发的流量。对于控制平面，SDN 需要一些机制或者协议来与设备进行通信，如应用层流量优化（application-layer traffic optimization，ALTO）协议[11]、路径计算单元（path computation element，PCE）[12]、OpenFlow 协议[13]等，本文主要运用的是 ALTO。

2.1.1 ALTO 协议

ALTO 协议是连接 ALTO 服务器以及 ALTO 客户端的接口。其中，ALTO 服务器承担収集网络拓扑信息的功能，ALTO 客户端则向 ALTO 服务器发送网络拓扑信息，并根据需要对拓扑信息进行筛选，以便为应用层服务。网络中的应用都能够根据自身的需求，依照 ALTO 协议向 ALTO 服务器请求网络信息。图 1 为 ALTO 服务的框架图。



图 1　ALTO 服务的框架图

ALTO 协议的服务主要包括映射服务、终端属性服务、映射过滤服务、终端开销服务等。以上所述的服务中，映射服务是核心服务，其余为可选服务。映射服务主要包括网络映射表和开销映射表。网络映射表中存储的是网络位置分组，不同分组表示不同的子网或者城域网。开销映射表反映了路由间的开销。ALTO 服务器根据路由开销确定不同网络的优先级，这样可以方便的修改网络资源消费模式。路由开销包括路由的跳数以及成本等。

ALTO 协议的设计主要是用来优化应用层的流量服务，网络应用可以依据这些服务获取网络信息，实现节点的非随机性选择，而网络信息将按照不同的应用需求在最大化带宽资源、最小化节点开销等方面实现平衡。在网络中使用 ALTO 协议所包含的服务和接口可以实现快速的网络优化，ALTO 协议的客户端到服务器的网络结构则可用来进行网络信息的传输和收集。

2.1.2 基于 SDN 的 ALTO

对于目前的 ALTO 架构来说，由于其包含有网络信息的收集，这就使得 ALTO 的计算开销增大，ALTO 也就具有较高的

复杂度。而对于 SDN 而言，信息收集的自动化是其主要特征。ALTO 的核心思想是向上层应用开放应用接口，将 ALTO 与 SDN 联合使用可以实现流量的优化。本节用控制器代替客户端，建立控制器和服务器间的通信模型，对比控制器的使用下 ALTO 中客户端、服务器之间的网络信息流量变化。

在使用控制器的融合 ALTO 架构中，由于 SDN 不同域的划分，协议收集的信息往往是来自某个特定 SDN 域而不是关于全局的信息（所有 SDN 域）。因此，为了使 ALTO 协议能够适配 SND 划分域，需要对其进行扩展。

2.2 网络功能虚拟化

网络功能虚拟化（NFV）提供了一种在一个虚拟化的基础设施中设计、部署和管理网络功能和服务的新方法。NFV 从专用硬件设备中解耦网络功能，如防火墙、入侵检测系统、负载均衡、网络地址翻译和缓存。设计它是为了部署网络组件，以支持虚拟化基础结构，包括虚拟机、服务器、存储等。其基本思想是由欧洲电信标准协会的团体于 2012 年提出的。

NFV 是 SDN 的一种互补方法。虽然它们均用于管理网络，但其依靠不同的方法。SDN 分离控制和转发平面以提供网络的集中视图，但是 NFV 专注于在独特的网络配置基础设施上实现网络服务，利用 SDN 实现 NFV 基础设施具有其固有的优势。

2.3 内存中的键值存储

内存中的键值存储（IMKVS）是像哈希映射一样运转的基本软件，它通过密钥映射内容，能够在缓存层内执行操作或存储嵌套的数据结构，但是其主要功能是键值关系。在应用程序中可以执行的基本命令如下所述：

void set（key，value）：询问特定的服务器以存储给定密钥识别的给定值。

value get（key）：检索在特定服务器中存储的由给定密钥识别的值。如果没有与密钥相关的值，则返回空。

void delete（key）：删除特定服务器给定密钥识别的值。

在今天的数据中心中，IMKVS 大多用来作为缓存层，该缓存层拥有复杂和耗时的任务，如大量的数据库查询结果。需要注意的是，IMKVS 是今天数据中心应用程序的关键部分，然而这些系统需要能够提供大量的旨在服务用户请求的应用程序请求。按照上面所描述的方法，最简单的用户请求可能会触发多个到缓存层的请求，这使得这些缓存随着时间的推移而被大量访问。

2.4 一致性哈希

Web 缓存的有限性促使了一致性哈希（CH）的出现。使用一致性哈希和 mod 运算可以将对象集均匀地分配到缓存服务器。令 O 为任意一个对象，n 为可用服务器的数量。则式（1）表示将对象 O 分配到服务器中：

$$i = hash(O) \bmod n \tag{1}$$

CH 的核心思想是将对象集合均匀分配到服务器集合中。当添加一个新的服务器到缓存层时，临近的对象与其共享，同时，当移除发生时，服务器的对象与其邻近共享。这种方法只会引起缓存层中局部变化，避免了服务事件上的热点或总对象重新分配。CH 可以采用的方法是通过使用对象和服务器上的一致性哈希映射对象到服务器范围，选择这些服务器之一来储存数据。算法 1 为基本算法步骤，其中 C_s 为服务器集，O 为对象，H 为哈希函数。

算法 1：一致性哈希

初始化循环表 L；
对于 C_s 中的每一个服务器 S，执行：
　$H_{sv} := H(S)$；
　$T := \{h = H_{sv}, s = S\}$；
　将 T 嵌入到 L；
结束
基于 $T.h$ 值分类 L；
$H_{ov} := H(O)$；
如果 L 包含任意 $T.h = H_{ov}$，则返回 $T.s$；
结束
否则
　将 H_{ov} 以排序方式嵌入到 L，比较 H_{ov} 和 $T.s$ 的值；
　返回 $T.s$ 值最右边的 H_{ov}；
结束

根据算法 1，图 2 显示了如何将两个随机对象映射到某些缓存服务器。其中 O_1 将被映射到 S_2，O_2 将被映射到 S_3。如果将 S_3 从缓存层中移除，则 O_2 将被映射到 S_1 或 O_4，因为它们是当前 S_3 的近邻。如果添加 S_5 到环，则它将被置于 S_2 和 S_4 之间，它们持有的对象将与 S_5 共享。

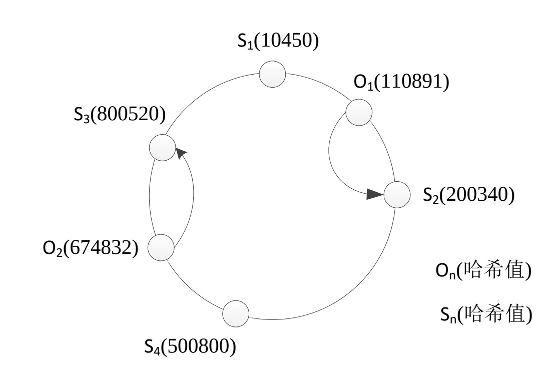


图 2　CH 环上重叠的两个对象

尽管 CH 穿过几个服务器在对象分布中具有良好的结果，但需要注意的是，它并没有把一切都考虑进去，这会引起网络热点。为了服务于用户和应用程序的复杂请求，数据中心会处理大量的对象，所以如果要降低将要创建的网络热点的概率，算法必须考虑网络索引。

3 两相负载平衡器

本章介绍了两相负载平衡机制，其旨在提高 IMKVS 缓存性能。提出的解决策略包含两个阶段，第一阶段设计通用的 SDN 负载平衡器的模块，以运行不同的负载平衡算法；第二阶

段是基于 IMKVS 的专业化缓存，可以实现通信管理和数据复制，可将其部署于非结构化网络之上，旨在代替 CH 算法。

在两个阶段中都会实现 VNF 功能。第一阶段负载平衡器部署虚拟化网络功能管理器（VNFM）模块，第二阶段负载平衡器部署 acfVNF 模块。所有信息通过分布式哈希表覆盖基础设施共享于所有 VNFM 和 VNF 模块之间。分发负载平衡器任务的原因是为本文提出的解决策略提供可扩展性，以缓解 SDN 控制器中的瓶颈。

如果在数据包到达的地方，交换机中没有配置路径，则客户端将命令发送给 IMKVS 服务器。交换机访问转发数据包的 SDN 控制器。一旦请求到达控制器，它将调用 VNFM 模块来检查高速缓存重定向虚拟机的当前状态，其目的是为了转发此请求到负载最少的 VM 实例，从而配置了请求交换机。这一步称作第一阶段负载平衡。

第二阶段负载平衡主要负载两项任务：平衡通过几个缓存服务器的负载；复制对象，以减轻持有流行对象的服务器的负载。它可能有多个执行负载平衡器的 VNF，从而使得本文提出的解决策略具有可扩展性，同时，使得它容易部署更多的缓存。这些 VMs 需要共享它们的内部转发状态以确保添加和移除 VMs 不会引起数据不一致。图 3 为本文提出的两相负载平衡器序列图。

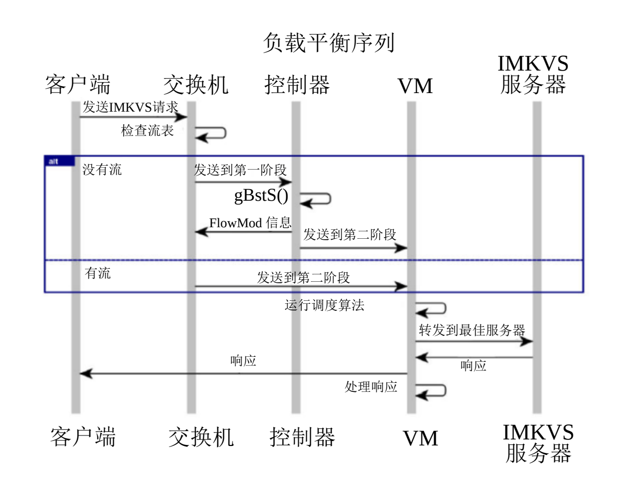


图 3　负载平衡序列图

假定一个客户端开启了到服务器 S_1 的请求，同时，这个服务器具有一个通过 TCP 处理连接的 IMKVS 实例。第一件必须发生的事情是三次数据交换，它包含了主机之间的协商，这里需要交换三个数据包：SYN、SYN+ACK 和 ACK。C_1 和 S_1 均建立了此连接。所有这些数据包需要到达对方，从而迫使交换机和控制器在没有干涉的情况下转发这些数据包。这表明本文算法是分布式的 VNF 负载平衡。当数据包到达交换机时，流表中将没有条目与之匹配，这将使得交换机发送 PacketIn 信息到控制器。一旦该数据包到达该控制器，则它必发送具有原始数据包目的地的 PacketOut 信息。在 C_1 和 S_1 建立连接之后，控制器可以分析额外的数据包以提取命令密钥。

这个过程中，将检查数据包的任务移动到 VNF 模块至关重要，这就涉及到客户端和服务器的连接。在本文提出的解决策略中，当客户端打开到服务器的连接时，第一阶段负载平衡器 VNFM 将转发数据包到第二阶段负载平衡器的 VNF 模块。这时候，为了将该请求转发给最佳的 IMKVS 服务器，第二阶段负载平衡器必须建立与客户端的直接连接并开始听从命令。

3.1　第一阶段负载平衡器

第一阶段的主要目的是将所有 IMKVS 的请求数据包复制到第二阶段负载平衡器，从而避免这些数据包通过第一阶段负载平衡器而被不均匀地重定向。由于缓存重定向器有很多 VMs，为了优化资源使用，请求通过 VMs 均匀到达很重要。对于本文所提出的 VNF，网络和 VMs 负载都很重要，它们是以同时考虑了两个条目的多项索引为特征的负载。

VM 要考虑的参数有 CPU 空闲率 (P_c)、RAM 空闲率 (P_r) 以及网络空闲率 (P_n)；网络设备要考虑的参数有可用宽带、传输错误率以及时延。

每个 VM 必须有一个代理来收集特定的数据并发送到 VNFM 网络控制器，它将只存储最后收到的数据。由于控制器是集中的且有通信通道到所有的网络设备以收集来自它们的数据，所以它必须在指定的时间间隔内请求有关其内部状态的每个设备，它还只存储最后的数据。从 VM 的角度出发，控制器是被动的，而从交换机的角度出发，控制器是主动的，可得负载 L_{VM} 的公式为

$$(P_n)L_{VM}=(100-\sqrt[3]{P_c\times P_r\times P_n})\times((t_c-t_a)\times f_{tp}+1\quad(2)$$

其中，t_c 为当前时间戳；t_a 为数据到达时刻的时间截；f_{tp} 为[0,1]间隔内的时间惩罚因子。

网络负载由来自客户端和 VMs 之间的链路负载所组成。令 P 为链路 L 的集合，n 为 P 的规模；可以如下式（3）建模路径负载 L_{net}。

$$L_{net}=\sum_{x=1}^{n}\left(100-(b-e)+\frac{100\times t}{t_{\max}}\right)\tag{3}$$

其中 b 为 L_x 可用宽带，e 为传输误差，t 为时延，$t_{\max}$ 为最大可接受时延。

算法 2：gBstSrv 程序

$t:=$ 全局负载阈值；

$L:=[]$；

对于 S 中的每个服务器 s，执行

　获得 s 的 L_{Net}；

　获得 s 的 L_{VM}；

　将 $\{srv=s,\ln et=L_{Net},lvm=L_{VM}\}$ 添加到 L；

结束

按 ASC 阶通过 $lnet$ 和 lvm 分类 L；

从 L 获得第一个元素 f；

$lnet_t:=f.lnet+t$；

$lvm_t := f.lvm + t$ 返回 $T.s$ 值最右边的 H_{ov}；

从 L 中获得第一个元素 nt，其 $lnet$ 和 lvm 值分别小于或等于 $lnet_t$ 和 lvm_t；

从集合 $(f+nt)$ 中返回随机数 srv；

控制器同时具有 VM 和网络负载后，则使用算法 2，其中，S 为可选服务器，s 为最佳服务器。它可以将请求转发到负载较少的服务器。它使用全局负载阈值 t 来收集一组负载较少的服务器而不只是负载多的服务器。这种方法旨在避免负载不平衡。

3.2　第二阶段负载平衡器

第二阶段负载平衡器有四个组分组成，如图 4 所示。箭头表示组分 A 取决于 B（A→B），内部-VM 连接表示网络通信。C 池和 S 池分别为到客户端和服务器的连接池。调度和复制组件都是最重要的。第一个负责根据负载平衡策略来处理和转发从客户端到服务器的请求。第二个能够获得最流行的缓存对象，为了避免网络中的热点，它会评估点击量并通过缓存层创建副本。索引负载通过映射对象的密钥到一组 IMKVS 服务器，将有关每个对象位置的、更新的分布式信息维护到缓存层。

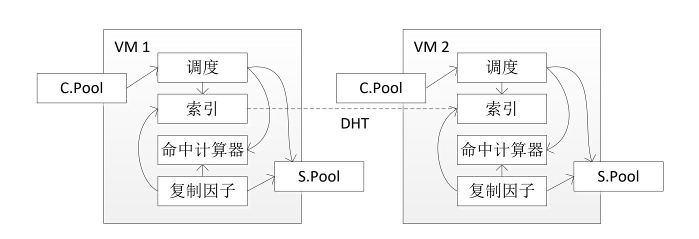


图 4　第二阶段负载平衡器的体系结构

3.2.1 需求

设计这个模块是为了通过平均分布穿过缓存服务器的 IMKVS 请求来处理 IMKVS 请求。通过这种方式，软件必须知道具体的 IMKVS 协议以确保它将与所有可用的命令兼容。

置位指令：当一个客户端试图将一个对象存储到缓存层中时，它将使用置位指令。

其他指令：所有其他指令假设所请求的对象已被储存于缓存中。此时，VM 只需要检查储存对象的服务器，并将请求转发给这些服务器之一。如果目标对象没有被存储在任何地方，则将自己的 VM 返回给请求者客户端，从而节约了资源。

为了完成以上所描述的要求，有必要建立一个索引，该索引用以知道对象存储在哪里。由于每个对象都被映射到一个唯一的键，所以它有可能使用哈希表，该表会将键 k 映射到服务器 S。然而，它不可能使用基本哈希表来完成这个，因为 VM 可能有多个实例，同时其键不转发到达特定的 VM 请求。于是，为了满足这个要求，这个索引必须与 VM 的共享，同时使用 DHT 完成索引。

3.2.2 索引

在 DHT 中，覆盖网中的每个节点负责保存表的一部分。这些分布式数据结构利用算法来提供查找操作。DHT 有能力处理数以百计的请求[14]。选择 DHT 而非独特集中式哈希表的主要原因是为了避免单点故障。实现 DHTs 的协议有多个，出于简单性和实施设施，本文选择 Chord 协议[15]。

Chord 在对数时间内提供查找操作，它需要 $(\log N)$ 信息来找到任意对象，其中 N 为网络中的节点数。假如每个 VM 每秒可以处理数据中心中的 500 个请求，且该数据中心每秒具有 10^6 个 IMKVS 请求，则整个 VNF 至少需要 2000 个服务器来处理这些请求。这意味着，在这样简单的环境中，Chord 将只需要 11 条信息来找到索引值。。假设两个 VMs 之间有 W 个交换机，则对于单一 Chord 查找，至少应有 $2\times W\times N$ 个数据包遍历于网络。

为了提高查找时间，它利用两个数据结构实施索引：内部索引和外部索引。内部索引为 NFV 实例内部的索引，外部索引为与所有 NFV 索引共享的实际 DHT。使用这个数据结构的原因是为了减少每个 IMKVS 请求的查找时间和网络流量。所以，内部和外部索引储存的对象包含了映射服务器 $S(k\rightarrow s)$ 的任意键 k。

对于索引上每个传入的查找请求，它首先实施内部索引查找，如果没发现结果，则检查共享的 DHT 外部索引。一旦外部索引查找返回请求值，则将其复制到将来查找请求的内部索引。当执行插入或删除请求时，软件将其同时发送到内部或外部索引。在 DHT 之前使用局部哈希表是为了减少查找成本，因为内部索引中的查找时间为 $O(1)$，而 DHT 路由表中为 $O(\log N)$，其中 n 为节点数。

3.2.3 调度程序

第二阶段负载平衡器的主要组分为调度程序，调度程序包括中央逻辑，它负责根据负载平衡策略来处理和转发从客户端到服务器的请求[16]。本节将介绍当请求进入 VM 实例时该程序的概述。

当请求到达 VM 实例时，调度程序检查由到任意服务器的客户端发送的指令。调度程序执行的第一步是将键提取到指令参考的地方。如果有多个数据包，则有必要读取前 P 个数据包来提取键，这一步忽略对来自这一指令的进一步数据包的检查。一旦该调度程序同时具有键和指令，即设置、获取、mget 或删除，则它可以根据指令类型执行特别行动。算法 3 将根据指令得以执行，其中 C 表示指令，　k 表示键。

算法 3：调度程序的设置算法

$S := \text{index.get}(k)$；

如果 S 不是空的，则

　对于 S 中的每个服务器，执行

　　打开新的多线程，并将 C 转发到 s；

　　结束

　当第一个多线程完成转发时，将其响应返回到请求者客户端；

结束；

否则

　$s := gBstSrv()$；

转发 C 到 s，并将其响应返回到请求者客户端；
索引 $\mathrm{put}(k.s)$；
结束

算法 3 和 4 为调度程序的重要算法。其基本思想是：通过查阅索引，检查内容储存在哪里；如果内容不存储在任何地方，则基于其电流负载选择一个可用的服务器。算法 4 可以从目标服务器中接收丢失的信息。

算法 4：调度程序的获得算法

$S := \mathrm{index.get}(k)$；
如果 S 不是空的，则
　$s := gBstSrv()$；
　转发 C 到 s 并获得 $resp$；
　返回 r 到客户端；
　如果 $resp$ 为错过，则
　　从索引中删除 k；
　结束
否则
　$GlobalHitConter+=1$；
　$KeysHitConter[k]+=1$；
　结束
结束
否则
　返回错过到客户端；
结束

算法 5 将一个 mget 请求分割为多个获得请求。然而，当一个服务器处理每个请求的大量键时，其处理可能是一个瓶颈，因为每个子请求是并行执行的，所以完成整个 mget 指令的时间将取决于网络条件。因为缓存分布在许多服务器之中，所以多个获得指令将会给出更优的性能。

算法 5：调度程序的 mget 算法

对于 K 中的键 k
　打开一个新的多线程并调用 $get(C,k)$；
结束

算法 6 旨在通过避免持有请求键的服务器之间的通信来减少请求时延。

算法 6：调度程序的删除算法

从索引中删除 k；
发送删除消息到请求者客户端；

需要注意的是，对于第一阶段负载平衡器，负载定义 L_{VM} 和 L_{net} 参考 VNFVMs，而对于第二阶段负载平衡器的调度程序，它们参考 IMKVS 服务器。在两相负载平衡器中，gBstSrv 调度程序会使用这些参数。与第一阶段负载平衡器相类似，会将来自服务器和网络的负载通知给所用的 VM 实例，但是在网络负载情况中，VNFM 询问 SDN 控制器的当前网络负载，而不是直接询问交换机。本文提出的体系结构注意过度监测，因为它可以影响服务器的性能和解决策略的扩展性。

3.2.4 复制

算法 7　复制算法

获得电流 VM 负载 L_{VM}；
如果 $L_{VM} \le L_{\max}$，则
　$P := []$；
　$g_{hc} := GlobalHitCounter$；
　对于每个命中计数器中的每个 k_{hc}，执行
　　$p := \dfrac{k_{hc} \times 100}{g_{hc}}$；
　　如果 $p \ge R_t$，则
　　　将 $\{key = k_{hc}.key, pc = p\}$ 添加到 P；
　　结束
　结束
　将 $GlobalHitCounter$ 和 $KeysHitCounter$ 重新设置为 0；
　$S :=$ 通过其负载分类的所有 IMKVS 服务器；
　对于 P 中的每个 p，执行
　　从索引中获得 k 参考的服务器集 S_k；
　　$S_r := S - S_k$；
　　$c := p.pc \times R_f$；
　　复制前 c 个 S_r 服务器上的 $p.$ 键；
　结束
结束

算法 7 描述了每个 VM 如何部署复制机制。为了检索到普及度超过给定复制阈值即 R_t 的键，该算法会检查计数器，并将对象复制到 $p \times R_f$ 服务器，R_f 为复制因子，p 为对象的普及率。

4　仿真分析

在 Java atop 控制器中实施第一阶段负载平衡器，使用连接处理的 Netty 构架在 Java 中实施第二阶段负载平衡器。每个 VM 都有一个对应于 VNF 实例的可追捕的.jar 菜单。所有图表使用 35 个样本尺寸，设置 95%的置信区间。在每次实验执行中，在所有链路中都设置有数据包丢失从 0 到 1ms 和从 0 到 3%的随机时延。

4.1　指令执行时间的比较

图 5 比较了每个 IMKVS 指令的执行时间。在这个实验中，配置有 10 个客户端来对 10 个服务器进行每秒 100 次的操作。为了测量本文提出解决策略的影响，本文只使用一个调度 VM 实例。结果表明，当与 CH 进行比较时，在所有的指令中，本文提出的解决策略提高了 IMKVS 请求的性能。在 mget 指令上可以看到最显著的改进，在该 mget 指令中，时间减少了 18%。此外，删除指令的时间提高了 45%。

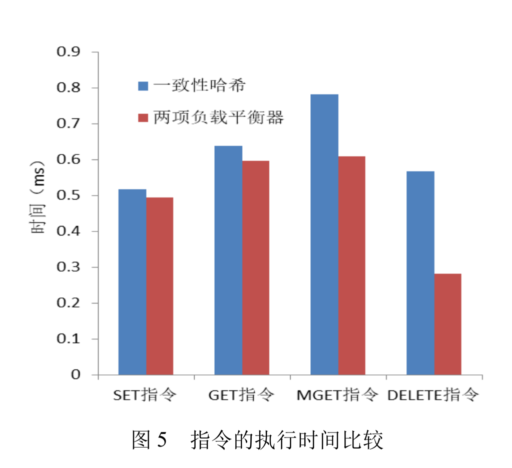


图 5　指令的执行时间比较

4.2　多个 VMs 的 mget 指令的执行时间

图 6 对第二阶段负载平衡器不同数量的 VM 实例的 mget 指令的执行时间进行了比较。结果显示，高 op/s 率的较低量的 VMs 会引起性能退化，从而改善了结果，这是因为第一阶段负载平衡器转发了有 VMs 负载的请求。此外，需要注意的是，预留空间不会提高 mget 指令的整体性能，但会引起时间的小幅提升，这是由于 DHT 共享索引上查找操作的增加导致的。

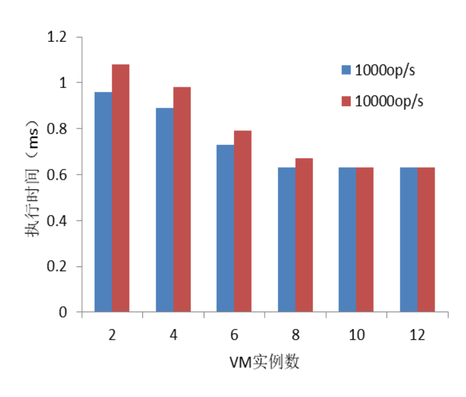


图 6　多个 VMs 的 mget 指令的执行时间

4.3　服务器和网络负载比较

图 7 为使用 CH 和两相负载平衡器时，网络的平均负载图。为了确保在相等的通信状态下执行两次测量，本文只生成对象及其键一次。结果表明，由于存储内容时选择负载较少的服务器的容量，本文提出的负载平衡器将服务器和网络的负载分别减少了 24%和 7%，具有较好的性能。

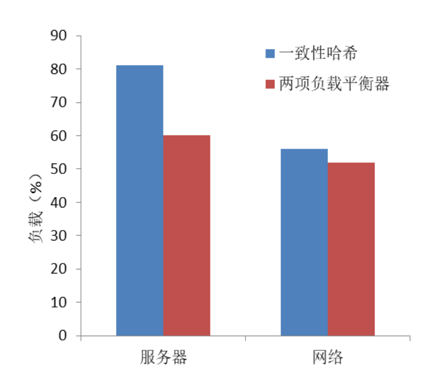


图 7　服务器和网络负载比较

4.4　服务器和网络能耗比较

图 8 为使用 CH 和两相负载平衡器时，网络的平均能耗图。本次仿真同样只生成对象及其键一次。相比于 CH 算法，本文算法的指令执行时间和网络负载都有所减小，所以本文提出的负载平衡器的能耗也有所降低。

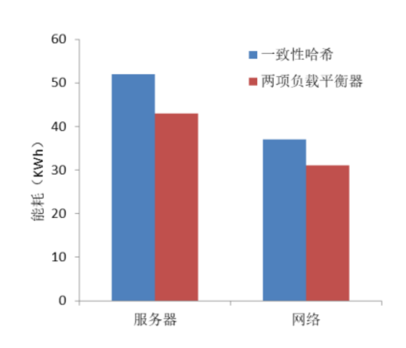


图 8　服务器和网络能耗比较

5　结束语

本文为 IMKVS 系统提出了两相负载平衡器，它将用于数据中心的分布式缓存中。本文提出的解决策略旨在以一种可扩张且容易部署的方式减少服务器和网络中的负载。在第一阶段模块中，仿真结果表明，相比于传统的网络转发，网络负载平衡得以提高，同时有助于减少分布式服务器处的负载。在第二个阶段模块中，仿真结果表明， 本文提出的解决策略优于 CH，同时减少了执行 IMKVS 系统基本指令的时间。本文提出的解决策略有能力提供更好的资源使用、减少操作成本并提高用户经验。

参考文献：

[1] 张伯阳，张晓，李阿妮，等. 云存储系统可扩展性评测研究 [J]. 计算机应用研究, 2017, 34 (7): 1957-1961.

[2] Esposito C, Ficco M, Palmieri F, et al. Smart cloud storage service selection based on fuzzy logic, theory of evidence and game theory [J]. IEEE Trans on Computers, 2016, 65 (8): 2348-2362.

[3] Yu Y, Man H A, Mu Y, et al. Enhanced privacy of a remote data integrity-checking protocol for secure cloud storage [J]. International Journal of Information Security, 2015, 14 (4): 307-318.

[4] Linden S V D, Rabe A, Held M, et al. The EnMAP-Box: a toolbox and application programming interface for EnMAP data processing [J]. Remote Sensing, 2015, 7 (9): 112-119.

[5] 胡丽聪，徐雅静，徐惠民. 基于动态反馈的一致性哈希负载均衡算法 [J]. 微电子学与计算机, 2012, 29 (1): 177-180.

[6] 刘鋆，虞红芳. 基于资源拆分的虚拟网络功能服务链映射算法 [J]. 计算机应用研究, 2016, 33 (8): 2440-2445.

[7] 杨沛霖，吕光宏，张团利. 关于软件定义网络可扩展性的综述 [J]. 计算机应用研究, 2016, 33 (4): 967-972.

[8] Wang R, Butnariu D, Rexford J. OpenFlow-based server load balancing

gone wild [C]// Proc of USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services. [S. l.] : USENIX Association, 2011.

[9] Handigol N, Seetharaman S, Flajslik M, et al. Plug-n-serve: load-balancing web traffic using OpenFlow [C]// Proc of ACM SIGCOMM Conference. 2009.

[10] Zhang K, Wang K, Yuan Y, et al. Mega-KV: a case for GPUs to maximize the throughput of in-memory key-value stores [J]. Proceedings of the VLDB Endowment, 2015, 8 (11): 1226-1237.

[11] 张伟，雷为民，李广野，等. 基于应用层流量优化的中继路径选择方案 [J]. 计算机工程与科学, 2014, 36 (10): 1880-1887.

[12] Cugini F, Meloni G, Paolucci F, et al. Demonstration of flexible optical network based on path computation element [J]. Journal of Lightwave Technology, 2012, 30 (5): 727-733.

[13] 左青云，陈鸣，丁科，等. OpenFlow 网络冗余控制报文消除机制研究 [J]. 计算机研究与发展, 2014, 51 (11): 2448-2457.

[14] Paik J H, Dong H L. Scalable signaling protocol for Web real-time communication based on a distributed hash table [J]. Computer Communications, 2015, 70 (C): 28-39.

[15] Prasanna P, Krishna M, Subramanyam M V, et al. Performance analysis of chord protocol for peer to peer overlay topology in wireless mesh network [J]. International Journal of Computer Applications, 2013, 65 (13): 49-52.

[16] 曲桦，赵季红，樊斌，等. 软件定义网络中应用蚁群优化的负载均衡算法 [J]. 北京邮电大学学报, 2017, 40 (3): 51-55.